

# MONKES MANUAL (DRAFT)

Francisco Javier Escoto López



<b>1</b>	<b>Installation and required libraries</b>	<b>3</b>
1.1	Fortran compiler . . . . .	3
1.2	NetCDF library . . . . .	3
1.3	LAPACK library . . . . .	3
<b>2</b>	<b>How to use MONKES</b>	<b>3</b>
2.1	Magnetic configuration input . . . . .	3
2.2	Monoenergetic database input . . . . .	3
2.3	Running the executable with SLURM . . . . .	3
2.4	Monoenergetic database output . . . . .	3
<b>3</b>	<b>Algorithm implementation</b>	<b>3</b>
3.1	User level . . . . .	3
3.2	Developer level: main routines . . . . .	7
3.3	Developer level: libraries . . . . .	7
<b>4</b>	<b>Application Programming Interface (API)</b>	<b>7</b>

# 1 Installation and required libraries

In this section it is explained how to install MONKES.

## 1.1 Fortran compiler

MONKES is written in Fortran language and thus requires having an installed Fortran compiler. This compiler must be compatible with LAPACK and NetCDF libraries. One possible compiler is the Intel Fortran compiler.

## 1.2 NetCDF library

In order to read BOOZER\_XFORM output files, MONKES needs to use the NetCDF library. An example of the minimal required libraries for running MONKES in a UNIX machine is displayed below. Modules 1) and 2) are the NetCDF library compatible with the Fortran compiler. Module 3) is the Fortran compiler version, in this case intel17/17.0.8.

Currently Loaded Modules:

- 1) netcdf-fortran-4.4.4-intel-17.0.8-4me7upi
- 2) netcdf-4.6.1-intel-17.0.8-i5cn5xw
- 3) intel17/17.0.8

## 1.3 LAPACK library

For using LAPACK library there are two options. One can use the static or the dynamic version of the library. In listing 7, the dynamic case is illustrated.

```
# *** Compiler selection
f90comp=gfortran
NETCDF_DIR=/usr/include/

FFLAGS=-I$(NETCDF_DIR) -mmodel=large -O2 -freal-4-real-8 -ffree-line-length-512
LFLAGS=-llapack -lnetcdf -lnetcdf -lblas
```

Listing 1: Makefile

The variable LFLAGS includes the linking of LAPACK libraries. Specifically, those flags which include the variable \$MKL\_HOME (variable containing the location of the dynamic library) set the instructions for where to find the library. The flag -mkl=parallel allows for LAPACK multithreading functionalities. In listing 3, it is shown how the monkes executable is generated.

```
main_monkes.x: $(objects)
    $(f90comp) -o main_monkes.x $(objects) ${LFLAGS} ${FFLAGS}
```

Listing 2: Makefile

# 2 How to use MONKES

## 2.1 Magnetic configuration input

## 2.2 Monoenergetic database input

## 2.3 Running the executable with SLURM

## 2.4 Monoenergetic database output

# 3 Algorithm implementation

## 3.1 User level

Listing 3: main\_monkes.f90

Listing 4: main\_monkes.f90

```

subroutine Monoenergetic_Database_Input
  integer, parameter :: N_nu_max = 500, N_E_r_max = 500
  integer, parameter :: M_theta_max = 500, M_zeta_max = 500, M_xi_max = 500
  real :: nu(N_nu_max), E_r(N_E_r_max)
  integer :: N_theta(M_theta_max), N_zeta(M_zeta_max) , N_xi(M_xi_max)
  integer :: N_nu, N_E_r, M_theta, M_zeta, M_xi, ierr
  namelist /parameters/N_theta, N_zeta, N_xi, nu, E_r
  namelist /parameters_DF/N_xi_DF, N_lambda
  integer, parameter :: M_xi_DF_max = 500, M_lambda_max = 500
  integer :: N_xi_DF(M_xi_DF_max), N_lambda(M_lambda_max)
  integer :: M_xi_DF, M_lambda
  logical :: Monoenergetic_lambda
  integer :: i, j, k, ii, jj, kk, iii, kkk, c0, c1, rate
  real :: t_clock, t_cpu, t0, t1, lambda_c
  real, allocatable :: F1(:,:,:), F3(:,:,:), Gamma_ij(:,:,:), lambda(:)
  real, allocatable :: dGamma_ij_dlambda(:,:,: )
  real, allocatable :: D(:,:,:,:), D_33_Sp(:,:,:,:)
  character(len=500) :: file_path

  call system_clock(count_rate=rate) ! Get click rate

  ! *** Read input parameters for monoenergetic database from
  ! "monkes_input.parameters" file.
  N_theta = -1 ; N_zeta = -1 ; N_xi = -1 ; nu = -1d14 ; E_r = -1d14
  open(1, file= "monkes_input.parameters", status="old")
  read(1, nml=parameters, iostat=ierr)
  close(1)

  ! Count number of collisionalities and radial electric field to be
  ! included in the database
  M_theta = count(N_theta > 0)
  M_zeta = count(N_zeta > 0)
  M_xi = count(N_xi > 1)
  N_nu = count(nu > 0) ; N_E_r = count(E_r /= -1d14)

  allocate( D( 3, 3, N_nu, N_E_r, M_theta, M_zeta, M_xi ) )
  allocate( D_33_Sp( N_nu, N_E_r, M_theta, M_zeta, M_xi ) )

  ! *** Read input parameters for distribution function and lambda from
  ! dependence of the monoenergetic coefficients from "monkes_input.parameters" file.
  N_xi_DF = -1 ; N_lambda = -1
  open(1, file= "monkes_input.parameters", status="old")
  read(1, nml=parameters_DF, iostat=ierr)
  close(1)
  Monoenergetic_lambda = (ierr == 0)

  if ( Monoenergetic_lambda ) then
    M_xi_DF = count(N_xi_DF > 0)
    M_lambda = count(N_lambda > 0)
  else
    M_xi_DF = 1 ; M_lambda = 1
    N_xi_DF = 2
  end if

  write(*,*) " *** Performing scan in collisionality and radial electric field "
  write(*,*) " nu/v [m^-1] "
  write(*,*) nu(1:N_nu)
  write(*,*)
  write(*,*) " E_r/v [kV s /m^2] "

```

```

write(*,*) E_r(1:N_E_r)
write(*,*)
write(*,*) " *** Scan done using the resolutions "
write(*,*) " N_theta (# points for poloidal angle) "
write(*,*) N_theta(1:M_theta)
write(*,*) " N_zeta (# points for toroidal angle) "
write(*,*) N_zeta(1:M_zeta)
write(*,*) " N_xi (# Legendre modes in pitch-angle cosine) "
write(*,*) N_xi(1:M_xi)
write(*,*)
if( Monoenergetic_lambda ) then
  write(*,*) " *** Extracting dependence of the monoenergetic coefficients "
  write(*,*) " on lambda for each case of the scan"
  write(*,*) " N_xi_DF "
  write(*,*) N_xi_DF(1:M_xi_DF)
  write(*,*) " N_lambda "
  write(*,*) N_lambda(1:M_lambda)
  write(*,*)
end if

where( mod(N_theta(1:M_theta),2) == 0 ) N_theta(1:M_theta) = N_theta(1:M_theta) + 1
where( mod(N_zeta(1:M_zeta),2) == 0 ) N_zeta(1:M_zeta) = N_zeta(1:M_zeta) + 1

write(*,*) " *** Monoenergetic Database "
write(*,*(9999A25)') " nu/v [m^-1]", " E_r/v [kV s /m^2]", &
  " N_theta ", " N_zeta ", " N_xi ", &
  " D_11 ", " D_31 ", &
  " D_13 ", " D_33 ", &
  " D_33_Spitzer ", &
  " Wall-clock time [s] ", &
  " CPU time [s] "

! Location for output
file_path = "monkes_Monoenergetic_Database.dat"
! Open output file and write header
open(21, file=trim(file_path))
write(21,*(9999A25)') " nu/v [m^-1]", " E_r/v [kV s /m^2]", &
  " N_theta ", " N_zeta ", " N_xi ", &
  " D_11 ", " D_31 ", &
  " D_13 ", " D_33 ", &
  " D_33_Spitzer ", &
  " Wall-clock time [s] ", &
  " CPU time [s] "

! OPEN (if necessary) monkes_Monoenergetic_lambda.dat
if( Monoenergetic_lambda ) then
open(31, file=trim("monkes_Monoenergetic_lambda.dat"))
write(31,*(9999A25)') " nu/v [m^-1]", " E_r/v [kV s /m^2]", &
  " N_theta ", " N_zeta ", " N_xi ", &
  " D_11 ", " D_31 ", &
  " D_13 ", " D_33 ", &
  " D_33_Spitzer ", &
  " M_xi ", " lambda ", " lambda_c ", &
  " d_11 ", " d_31 ", &
  " d_13 ", " d_33 ", &
  " d d_11 / d lambda ", " d d_31 / d lambda ", &
  " d d_13 / d lambda ", " d d_33 / d lambda "
endif
do j = 1, N_E_r ! Loop electric field value
  do i = 1, N_nu ! Loop collisionality value
    do kk = 1, M_xi ! Loop number of Legendre modes
      do ii = 1, M_theta ! Loop number of theta points
        do jj = 1, M_zeta ! Loop number of zeta points
          do k = 1, M_xi_DF ! Loop on number of modes extracted of the distribution
function
            do iii = 1, M_lambda

```

```

call system_clock(c0) ; call cpu_time(t0)
allocate( F1(0:N_theta(ii)-1, 0:N_zeta(jj)-1, 0:N_xi_DF(k) ) )
allocate( F3(0:N_theta(ii)-1, 0:N_zeta(jj)-1, 0:N_xi_DF(k) ) )

! Solve the DKE and extract N_xi_DF(k)+1 Legendre modes
call Solve_BTD_DKE_Legendre_DF( N_theta(ii),      &
                                N_zeta(jj),        &
                                N_xi(kk),          &
                                nu(i), E_r(j),      &
                                D(:, :, i, j, ii, jj, kk), &
                                D_33_Sp(i, j, ii, jj, kk), &
                                N_xi_DF(k), F1, F3 )

call system_clock(c1) ; call cpu_time(t1)
! Wall-clock time in seconds
t_clock = ( c1 - c0 ) * 1d0 / rate

! Writing results on terminal
write(*, '(9999e25.16)') nu(i), E_r(j), &
    real(N_theta(ii)), &
    real(N_zeta(jj)), &
    real(N_xi(kk)), &
    D(1,1,i,j,ii,jj,kk), &
    D(3,1,i,j,ii,jj,kk), &
    D(1,3,i,j,ii,jj,kk), &
    D(3,3,i,j,ii,jj,kk), &
    D_33_Sp(i,j,ii,jj,kk), &
    t_clock, t1-t0

! Writing results on "monkes_Monoenergetic_Database.dat"
write(21, '(9999e25.16)') nu(i), E_r(j), &
    real(N_theta(ii)), &
    real(N_zeta(jj)), &
    real(N_xi(kk)), &
    D(1,1,i,j,ii,jj,kk), &
    D(3,1,i,j,ii,jj,kk), &
    D(1,3,i,j,ii,jj,kk), &
    D(3,3,i,j,ii,jj,kk), &
    D_33_Sp(i,j,ii,jj,kk), &
    t_clock, t1-t0

flush(21)

if( Monoenergetic_lambda ) then

    allocate( lambda(0:N_lambda(iii)), Gamma_ij(3,3,0:N_lambda(iii)) )
    allocate( dGamma_ij_dlamba(3,3,0:N_lambda(iii)) )
    ! Call the routine that computes Dij as lambda functions
    call Monoenergetic_lambda_function( N_lambda(iii), &
                                        N_theta(ii), N_zeta(jj), &
                                        N_xi(kk), N_xi_DF(k), &
                                        F1, F3, &
                                        lambda, lambda_c, &
                                        Gamma_ij )

    call Grid_initialization( "lambda", lambda, 2 )

    dGamma_ij_dlamba(1,1,:) = Derivative( "lambda", Gamma_ij(1,1,:), 1 )
    dGamma_ij_dlamba(3,1,:) = Derivative( "lambda", Gamma_ij(3,1,:), 1 )
    dGamma_ij_dlamba(1,3,:) = Derivative( "lambda", Gamma_ij(1,3,:), 1 )
    dGamma_ij_dlamba(3,3,:) = Derivative( "lambda", Gamma_ij(3,3,:), 1 )

    ! Write Dij(lambda) in monkes_Monoenergetic_lambda.dat
    do kkk = 0, N_lambda(iii)

        write(31, '(9999e25.16)') nu(i), E_r(j), &
            real(N_theta(ii)), &

```

```

real(N_zeta(jj)), &
real(N_xi(kk)), &
D(1,1,i,j,ii,jj,kk), &
D(3,1,i,j,ii,jj,kk), &
D(1,3,i,j,ii,jj,kk), &
D(3,3,i,j,ii,jj,kk), &
D_33_Sp(i,j,ii,jj,kk), &
real(N_xi_DF(k)), &
lambda(kkk), lambda_c, &
Gamma_ij(1,1,kkk), &
Gamma_ij(3,1,kkk), &
Gamma_ij(1,3,kkk), &
Gamma_ij(3,3,kkk), &
dGamma_ij_dlamba(1,1,kkk), &
dGamma_ij_dlamba(3,1,kkk), &
dGamma_ij_dlamba(1,3,kkk), &
dGamma_ij_dlamba(3,3,kkk)

flush(31)
end do

deallocate( lambda, Gamma_ij, dGamma_ij_dlamba )
end if
deallocate( F1, F3 )

end do
end do
end do
end do
end do
end do
end do
close(21) ! close monkes_Monoenergetic_database.dat
! CLOSE (if necessary) monkes_Monoenergetic_lambda.dat
if( Monoenergetic_lambda ) close(31)

end subroutine

```

Listing 5: examples/API\_Example\_DKE\_BT\_D\_Solution\_Legendre.f90

The subroutine `Monoenergetic_Database_Scan` computes the monoenergetic database by looping in the different parameters. For this, it calls within the loop the subroutine `Solve_BT_DKE_Legendre`. What the routine `Solve_BT_DKE_Legendre` does is out of the scope of this section (see Developer section). The output is written in the file `monkes_Monoenergetic_Database.dat`

Listing 6: examples/API\_Example\_DKE\_BT\_D\_Solution\_Legendre.f90

## 3.2 Developer level: main routines

## 3.3 Developer level: libraries

# 4 Application Programming Interface (API)